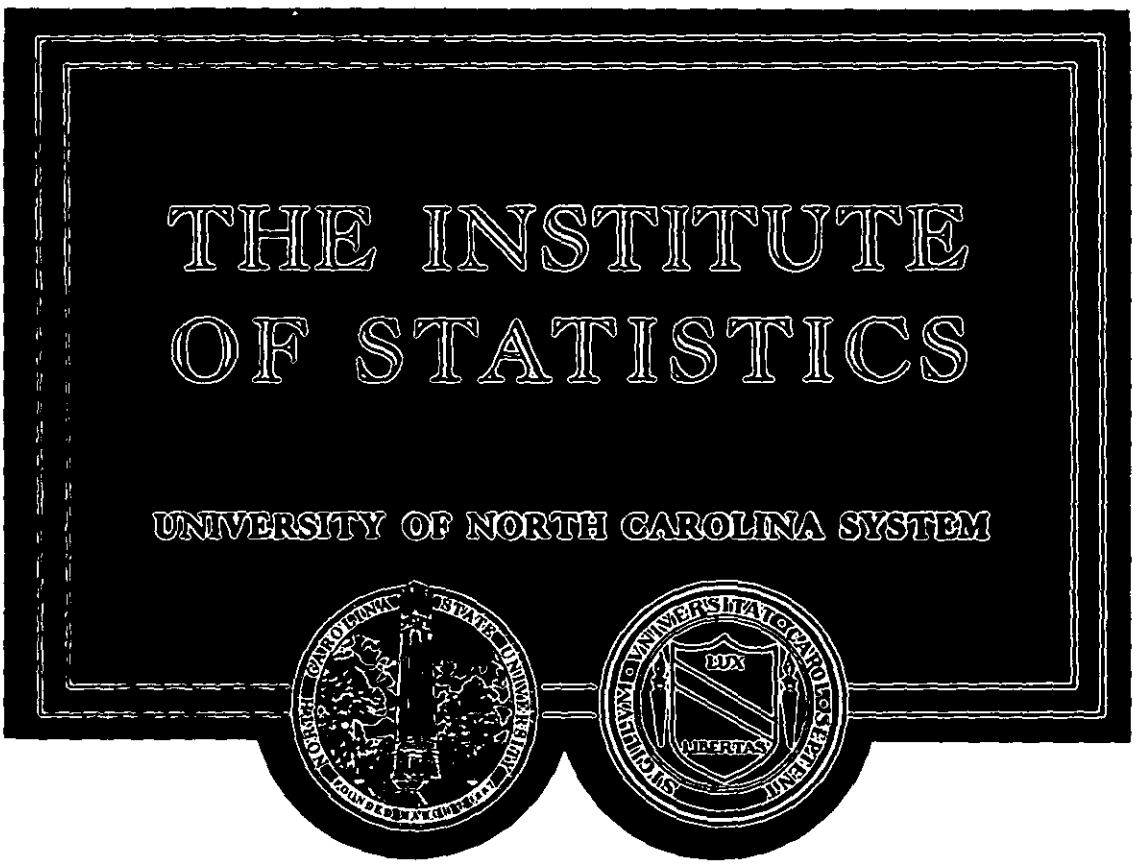


LENNS
a program to estimate the dominant Lyapunov exponent of noisy nonlinear systems from time series data

Stephen Ellner, Douglas W. Nychka and A. R. Gallant

Institute of Statistics Mimeo Series No. 2235
Biomathematics Series No. #39
September 1992

NORTH CAROLINA STATE UNIVERSITY
Raleigh, North Carolina

LENNS,
a program to estimate the dominant Lyapunov exponent of noisy nonlinear systems from time series data

Stephen Ellner*†, Douglas W. Nychka†, and A.R. Gallant†

(*)Biomathematics Graduate Program, and (†)Department of Statistics
North Carolina State University
Raleigh, NC 27695-8203, USA

This document may be cited as follows:
S. Ellner, D.W. Nychka, and A.R. Gallant. 1992. LENNS, a program to estimate the dominant Lyapunov exponent of noisy nonlinear systems from time series data. **Institute of Statistics Mimeo Series # 2235** (BMA Series # 39), Statistics Department, North Carolina State University, Raleigh NC 27695-8203.

1. INTRODUCTION

LENNS is a set of FORTRAN programs to estimate the dominant Lyapunov exponent of a noisy, nonlinear system from a time series of measurements, using methods described by Nychka et al. (1992). The fundamental assumption is that the data $\{x_t, t = 1, 2, ...N\}$ are generated by a nonlinear system of the form

$$x_t = f(x_{t-1}, x_{t-2}, \cdots x_{t-d}) + \sigma e_t$$

where $x_t \in \mathrm{R}^1$ and e_t is a series of independent, identically distributed random variables with zero mean and unit variance. LENNS estimates f by nonlinear regression, and uses the estimated map $\hat{f}$ and the data $\{x_t\}$ to produce an estimate of the dominant Lyapunov exponent. A positive Lyapunov exponent (for a bounded system) is an operational definition of "chaos", so one possible use of LENNS is to detect chaos in data from a noisy system. However, negative exponents are also meaningful, as discussed below.

LENNS has three distinguishing features relative to other available methods for estimating Lyapunov exponents from data:

1. The method in LENNS explicitly allows for *dynamic noise* (random perturbations to the dynamics), and the estimated Lyapunov exponent refers to the noisy system, rather than a hypothetical system from which noise has been removed. The method is valid in principle even for large dynamic noise ($\sigma \gg 0$). However the methods assume that *measurement errors* are negligibly small compared to the fluctuations in the data, i.e., $\mathrm{Var}(m_t) \ll Var(x_t)$ where m_j is the error in measuring x_j.

2. The map f is approximated by the input-output map of a feedforward neural network with a single layer of "hidden units" (analog neurons). The number of units determines the model complexity (i.e., a complicated $\hat{f}$ that comes close to interpolating the data, *versus* a simpler $\hat{f}$ that smooths the data). The program output lets the user choose the number of units based on based on generalized cross-validation or similar quantitative criteria for model selection.

3. The data can be fitted with a model that includes periodic forcing of known period, e.g. for data on biological populations affected by seasonal changes in environmental factors, or on a periodically stimulated oscillator.

Parameters are estimated by nonlinear least squares, which is time consuming due to the large number of local minima in the objective function. Consequently LENNS is limited in practice to short data sets, meaning roughly 500 or so data points on current workstations or 486 PC's. For larger data sets, it is probably preferable to use local approximations

(e.g., local polynomials as in Brown et al. (1991), or local splines as in McCaffrey et al. 1992).

In Section 2 we give some background on Lyapunov exponents and neural net time series models. If you're in the nonlinear dynamics game you can just skim it to get our definitions and notation. Section 3 describes how to compile and use the programs. Section 4 goes through an example and discusses the output.

The programs, and this user's manual (as a set of PostScript files) are in the files lenns.tar.Z (UNIX compressed tape archive) and lennsarc.exe (DOS self-extracting archive file). Those files are available via anonymous ftp at ccvr1.cc.ncsu.edu (128.109.212.20) in directory pub/arg/lenns. The software is provided at no charge for research purposes, "as is" without warranty of any kind, express or implied.

2. BACKGROUND

Traditionally the search for evidence of chaos in time series data has been based on deterministic nonlinear models, and has been considered a distinct alternative to stochastic modeling. However in many areas – e.g., economics, epidemiology, neurobiology, population biology – there is no evidence to justify the *a priori* assumption of strictly deterministic dynamics. A more comprehensive approach is to include both a nonlinear component describing the system's endogenous (internal) dynamics, and a stochastic term that accounts for perturbations by exogenous factors or random fluctuations in parameters affecting the system.

Casdagli (1992a) has shown that the method of reconstruction in time delay coordinates also applies to systems with forcing by an exogenous variable u_t. In general, the reconstructed dynamics take the form

$$x_t = f(X_{t-1}, U_{t-1}) \ , \qquad 1 \le t \le N \tag{1}$$

where $X_{t-1} = (x_{t-1}, x_{t-2}, \cdots x_{t-d})$ is the reconstructed state vector, and $U_t = (u_{t-1}, u_{t-2}, \cdots x_{t-K})$ is a vector of lagged inputs. Here we assume that the effect of the exogenous variables is additive:

$$x_t = f(X_{t-1}) + e_t, \qquad 1 \le t \le N \tag{2}$$

where $\{x_t\}$ are the data, f is an unknown function, and $\{e_t\}$ is a sequence of zero-mean, independent random perturbations. This won't be true in all cases, but may be a useful first approximation; for example most time-series models in population biology assume

that the log-transformed data satisfy (2). The data can then be represented as a time series of state vectors generated according to

$$X_t = F(X_{t-1}) + E_t \tag{3}$$

where $E_t = (e_t, 0, ..., 0)^{\mathrm{T}}$ and $F : \mathrm{R}^d \to \mathrm{R}^d$ is defined to reproduce the dynamics of (2). It is important to note that the random "noise" in (2) and (3) is not measurement error, but is an intrinsic part of the dynamics.

A useful operational definition of "chaos" for systems such as (2) is: bounded fluctuations in $\{x_t\}$ with *sensitive dependence on initial conditions* (e.g., Eckmann & Ruelle, 1985); that is, exponential divergence of trajectories with similar initial conditions. The dominant Lyapunov exponent, λ, quantifies the divergence rate and remains well-defined for noisy systems (Kifer, 1986). Thus the Lyapunov exponent can be used to quantify the degree to which a system is stable or chaotic, without presupposing that the system is deterministic.

Since $\{E_t\}$ are independent, the vector process $\{X_t\}$ is Markov, and we also assume that the process is strictly stationary and ergodic. The ergodic property insures that Lyapunov exponents can be expressed as a time average over a single trajectory.

Lyapunov exponents for stochastic systems. We use the standard definition of the dominant Lyapunov exponent λ for the system (3):

$$\lambda = \lim_{m \to \infty} \tfrac{1}{m} log \| J_m J_{m-1} \cdots J_1 \|, \tag{4}$$

where $J_t = DF(X_t)$, the Jacobian matrix of F at X_t, and $\| \cdot \|$ is the L_2 matrix norm, $\|A\|$ = (spectral radius of $A^{\mathrm{T}}A)^{1/2}$ (any matrix norm can be used but the L_2 norm is most common in the literature). In the setting described above, with a regularity condition on the Jacobians, the limit at (4) is constant with probability 1 (Kifer 1986). Thus λ gives the long-term rate of divergence between two trajectories that differ only in having a "small" difference in their initial conditions ("small" meaning "infinitesimally small", so that the variational system on the right hand side of (4) describes how the trajectories diverge). This definition of λ for stochastic systems is discussed more fully by Casdagli (1992a). Dynamical noise affects the distribution of X_t and consequently it can affect the stability of the system (e.g., move it into or out of chaos; Crutchfield et al. 1982). Therefore λ cannot be defined for (3) simply by deleting E_t and applying the definition for a deterministic system.

The sign of λ tells us whether the system's endogenous dynamics, represented by f

in equation (2), amplifies or dampens over time the effects of exogenous perturbations. A chaotic system ($\lambda > 0$) is a "noise amplifier": the effects of perturbations are compounded and cannot be ignored in predicting the future state of the system. A non-chaotic system is a "noise muffler": the effects of a perturbation are transient and asymptotically have no effect on the system's long-term dynamics.

Estimating Lyapunov Exponents with nonlinear regression. Approaches for estimating λ can be classified into *direct* methods and *regression* methods. Direct methods seek to find similar pairs of state vectors within the observed series and estimate how the subsequent trajectories diverge (Guckenheimer 1982, Wolf et al. 1985). This procedure is sensitive to dynamic noise, since the trajectories being compared will not have the same sequence of random shocks and will diverge due to the random component even if $\lambda < 0$, so inflated estimates of λ are obtained (Sayers 1990).

Regression methods generate estimates of λ through the intermediate step of estimating the map f and its derivatives. Let $\widehat{J}_t$ denote the estimate of the Jacobian matrix J_t obtained from the approximate map $\widehat{f}$, and $\widehat{T}_M = \widehat{J}_M \cdot \widehat{J}_{M-1} \cdots \widehat{J}_1$. The obvious estimate of λ is then

$$\widehat{\lambda}_N = \tfrac{1}{N}\, log\|\widehat{T}_N\| \tag{5}$$

McCaffrey et al. (1992) provide a formal proof that if $\widehat{J}$ is obtained from a consistent estimator of the system map f and its partial derivatives, a consistent estimate of λ can be obtained ("consistent" means that the estimated λ converges to the true value as the sample size increases), and Ellner et al. (1990) discuss the convergence rate. The estimate (5) is positively biassed (i.e.,$E(\widehat{\lambda}_N) \geq \lambda$ with equality only in exceptional cases), so LENNS calculates also the "QR" estimate suggested by Abarbanel (1992),

$$\widehat{\lambda}_{N,QR} = \tfrac{1}{N}\, log\|\widehat{T}_N v\|, \tag{6}$$

where v is a fixed vector of norm 1, usually $v = (1, 0, 0, ..., 0)^{\mathrm{T}}$. Since (5) equals the supremum of (6) over all vectors v of norm 1, $\widehat{\lambda}_{N,QR} \leq \widehat{\lambda}_N$. The difference between (5) and (6) is usually minor for 50 or more data points, but (6) has the smaller bias in our simulation studies and would be the preferred estimate to report. However, a formal consistency proof is not available for (6).

Originally linear regression was used to estimate the Jacobian matrices (Eckmann et al. 1986), but more sophisticated methods are now used. Brown et al. (1991) discuss methods for deterministic systems based on local polynomial regression, "local" meaning

that at each point x, a low-order polynomial is used to approximate f in a neighborhood around x.

Neural network models. For limited time series data on noisy systems, we have obtained the most reliable results when f is estimated using global nonlinear regression based on neural networks (McCaffrey et al. 1992, Nychka et al. 1992). "Neural networks" refers to a class of nonlinear models inspired by the neural architecture of the brain (e.g., McCulloch and Pitts 1943, IJCNN 1989). Statistical applications have mostly used feedforward, single hidden layer networks. The form of this model is

$$\text{(7)} \qquad \mathrm{f}(x_1, x_2, \ldots, x_d) = \beta_0 + \sum_{j=1}^{k} \beta_j \psi(X'\gamma_j + \mu_j)$$

where $\gamma_j = (\gamma_{j1}, \gamma_{j2}, \cdots \gamma_{jd})^{\mathrm{T}}$ and ψ is a univariate nonlinear "activation function", typically the logistic distribution function $\psi(u) = e^u/(1+e^u)$. LENNS uses a rational approximation to the logistic to reduce computation time. There are k(d+2)+ 1 free parameters in model (7).

Function approximation properties of neural networks have been studied by Hecht-Nielsen (1989), Hornick et al. (1989a), Cybenko (1989), Gallant and White (1991) and Barron (1991a). For time series modeling the network parameters $\{\beta_i, \gamma_i, \mu_i\}$ are estimated by nonlinear least squares or similar criteria. Some consistency results for least squares estimates are given by White (1989), and McCaffrey (1991) has given upper bounds on the convergence rate. Barron (1991a) has derived improved bounds for complexity-penalized network estimates that are restricted to a discrete set of parameter vectors. See Casdagli and Eubank (1992) for papers describing some successful applications of neural networks for modeling time series data.

Why nets? Because they work – meaning that the functions often used as "typical" examples of low-dimensional chaos (Hénon, Rössler, ...) are well approximated by neural nets. Why should this be? Barron's (1991b) results suggests that neural net models are especially suitable for functions dominated by low-frequency components. Let Γ_c be the class of functions f on R^d such that

$$\int |\omega|\, |\,\widetilde{\mathrm{f}}(\omega)|\, d\omega \leq c$$

where $\widetilde{\mathrm{f}}$ is the Fourier transform of f, and c is a fixed constant. Barron (1991b) shows that a complexity-penalized fitting criterion can achieve convergence rate

$$E\,\|f - \hat{f}_N\|^2 = \mathcal{O}\left(\tfrac{d}{N} log N\right)^{1/2}$$

for any f$\in \Gamma_c$, where N is the number of data points, $\hat{f}_N$ is the estimate based on N data points, and $\|f\|^2 = \int |f|^2 d\mu$ for some measure μ with compact support. For large d, this rate is better than the optimal rate for approximating arbitrary smooth functions (e.g., arbitrary functions in a Sobolev space) with a general-purpose nonparametric method such as splines. An intuitive interpretation of this result is that nets can "beat" general purpose nonparametric methods if the functions being estimated are high-dimensional but otherwise fairly simple.

Smoothing parameter selection. In practice, fitting a neural network model involves specifying the values of various "smoothing parameters" that control the complexity of the fitted model:

. *The number of units.* This determines whether the network with least-squares parameters will interpolate or "smooth" the data (which are appropriate if noise is absent or present, respectively).

2. *Reconstruction parameters.* One would like to use as few lagged variables as possible, in order to hold the number of fitted parameters to a minimum. A strategy based on the ideas of attractor reconstruction is to consider models of the form

$$x_{t+T_p} = f(x_t, x_{t-L}, x_{t-2L}, \cdots, x_{t-(d-1)L})$$

T_p is the prediction time, and L is usually called the "time-delay".

Choice of smoothing parameters is critical to obtaining accurate results. A model with too few units or too low an embedding dimension will not be able to approximate the dynamics; a model with too many parameters will fit the noise instead of only fitting the endogenous feedbacks.

One popular criterion for time series models is to select smoothing parameters to maximize prediction accuracy (e.g., Casdagli 1989, 1992b, Sugihara and May 1990). To avoid overfitting, prediction accuracy can be quantified by cross-validation: delete each data point (one at a time), fit the model to the reduced dataset, and then use the fitted model to predict the deleted point. Cross-validation is a standard approach for smoothing parameter selection in nonparametric regression (e.g., Wahba 1990) and has been suggested for choosing the embedding dimension d (Sugihara and May 1990, Cheng and Tong 1992). Similar ideas for "optimizing" attractor reconstruction have appeared several times in the chaos literature.

Cross-validation for smoothing parameter selection has a large amount of variability, occasionally yielding estimates that drastically undersmooth the data. In Nychka et al.

(1992) it was found that inflating the effective number of parameters in the generalized cross-validation (GCV) criterion avoids overfitting. This technique has also been reported to be useful for other estimators (Friedman 1990, Hastie and Tibshirani 1990, Friedman and Silverman 1989). For a model fitted to n data points, let p be the number of parameters (or "effective number of parameters", see Wahba 1990), and RMS the root mean square prediction error. The modified generalized cross-validation function is

$$V_c = \{RMS/(1 - p\frac{c}{n})\}^2 \quad ; \tag{8}$$

the standard GCV criterion is V_c with c= 1.

When a regression estimate is a linear function of the data (e.g., splines) there are usually shortcuts in computing V_c, but for nonlinear estimators such as neural nets such simplifications are not possible and cross-validation is impractical. One shortcut is to reserve part of the data strictly for evaluating the model's prediction accuracy (e.g., Casdagli 1989, Sugihara and May 1990), but this reduces the data available for parameter estimates. Consequently a number of criteria have been developed which approximate the cross-validated prediction accuracy based on the error variance of the fitted model adjusted by the number of fitted parameters. One such criterion, the Bayesian Information Criterion or BIC, has been found effective for fitting neural net models to data from noisy, nonlinear systems (Nychka et al. 1992, Granger and Teräsvirta 1992). The BIC criterion is

$$\text{BIC} = \tfrac{1}{2}\{1 + \log(2\pi) + 2log(RMS) + p\frac{logn}{n}\}. \tag{9}$$

However Granger and Teräsvirta (1992) observed that BIC is prone to overfit noisy *linear* data, so BIC is recommended only if linearity has been rejected based on some other criterion. In our experience V_c with $c = 2$ is less prone to overfit linear data.

Autocorrelated data create some additional pitfalls for smoothing parameter selection based on prediction accuracy. If successive values are highly correlated, the best one-step-ahead predictor is essentially a linear function of the most recent values, which ignores the long-term dynamics that are the actual object of interest. For this reason, cross-validation must consider prediction accuracy more than one step ahead. Another potential problem is that for high-dimensional models containing many irrelevant lags, the only data vectors close to X_t will be X_{t-1} and X_{t+1}. Thus a function estimate that gives most of its weight to X_{t-1} and X_{t+1} will do a good job of predicting the value at X_t; this favors complicated models that interpolate rather than smooth (e.g., a small bandwidth in kernel models). A straightforward "fix" is the one suggested by Theiler (1986) for comput-

ing the correlation dimension: omit temporal neighbors. "Leave one out" cross-validation should be replaced by "leave 2W+1 out": omit an additional W points before and after (in time) the point being to be predicted, with W large enough that there is no appreciable correlation between the point being predicted and the data used to fit the model.

3. INSTALLING & USING THE PROGRAMS

The program files are:

lenns.for	The main program and a few subroutines called only by the main. For a range of model specifications, the main estimates network parameters by nonlinear least squares, computes estimated Jacobians and uses them to compute the estimated Lyapunov exponent.
lenns.par	Parameter file. The entries in this file (explained below) control what happens on a particular run of the program.
objfun.for	Subroutines to compute the RMS prediction error and its gradient as a function of model parameters.
bfgsfmin.for	Subroutines for minimizing a multidimensional function with user-supplied gradient, for use in nonlinear least squares.
liapqr.for	Subroutines to calculate the Lyapunov exponent estimates.
dblas_le.for linpakle.for	Some BLAS and LINPACK routines
util_le.for	"Utility" routines (e.g., a random number generator).

Also provided are:

makefile	Instructions for UNIX or DOS "make" utilities
lenns1.par	Parameter file1 for the example discussed in section 4
*.asc	Example data files
cvmbed.for	Program to choose the time delay by ordinary cross-validation
cvmbed.par	Parameter file for cvmbed.for

Portability: The programs have been tested on a SUN SPARCstation IPX (f77 compiler) and a DOS 486 PC (Lahey f77L-EM/32 v.4 compiler). The programs are in orthodox FORTRAN77 with one exception: the system clock is used to seed the random number generator. In the SUN version this takes the form

```
iseed=time()
```

and for Lahey f77, **integer*4 function time()** is at the end of the lenns.for. For other compilers you can modify **function time()** or substitute a read from the keyboard.

UNIX systems: Get the file lenns.tar.Z (binary ftp) and issue the commands

```
uncompress lenns.tar.Z
tar xvf lenns.tar
make lenns.u
```

to uncompress the archive file; extract files and subdirectories from the archive; and compile the code to produce the executable file lenns.u . If "make" fails, try editing the makefile to suit your system, or combine all the *.for files into a single file and compile it. If optimized BLAS or LINPACK libraries are installed on your machine, the program will run faster if you edit the makefile to use those instead of compiling dblas_le.for and linpakle.for .

DOS systems: Get the file lennsarc.exe (binary ftp). This is a self-extracting DOS archive file. Issue the command

```
lennsarc
```

at the DOS prompt to extract the files into your current directory. The makefile provided works with Lahey f77L-EM/32 v.4 ; otherwise compile and link the source files listed above.

Running the program: You need to create a data file, edit the parameter file lenns.par, and run lenns.u (UNIX) or lenns.exe (DOS).

The data file must be an ASCII file listing the data values from first to last, one per line, in REAL format – so a data file that starts

```
7.0
86.0
99.0
6.0
```

is fine, but

```
7
86
99
6
```

will crash the program.

Here is the sample parameter file lenns1.par (the italics are not part of the file):

```
dmgm10.asc              data file name: data are read from this file
dmgm10.le               output file name: results are written to this file
125  18  1  1          nxmax  jt1  ltr  ifreq
1    6                  dmin    dmax <-- min & max # of lags
1    8   1              kmin    kmax  kcrit <-- min & max # of hidden units
1    3                  lmin    lmax <-- min & max time delay L
1   0.0                 Tp      alpha
0    12                 jforce jper
2.0                     cgcv
0.5                     *scale
.000001                 *tol1
.0000000005             *tol2
```

```
800   10000          *itmax1    *itmax2
50    250            *maxstep1  *maxstep2
0     .1             *ibrent    *fltol
```

nxmax: Number of data points to use. If nxmax is greater than than the number of data points in the input file, the entire data set will be used.

ltr: 1 or 0. If ltr= 1 the data are log-transformed.

ifreq: If ifreq= 1 every data value is read in; if ifreq= 2 only every other value is used; if ifreq= 3 only every third value is used, etc.

dmin, dmax; kmin, kmax, kcrit; lmin, lmax: The program loops over all combinations of d (#lags) and L (time delay) within the limits specified by these parameters. For each such combination it fits network models with k= *kmin*, kmin+1, kmin+2, ..., kmax units, but stopping once the criteria BIC and/or V_c have failed to improve for two successive k values. kcrit determines which criterion is used: BIC if kcrit= 0, V_c if kcrit= 1, and both if kcrit= 2.

t_1, T_p, α: The program fits a model of the form

$$x(t+T_p) \;=\; \alpha x(t) + f(\,x(t), x(t-L), ...) + \text{noise.}$$

T_p is the "prediction time". If $T_p < 0$ in the parameter file then $T_p = L$ is used, and T_p changes as L changes; otherwise T_p is held fixed.

$t_1 + 1$ is the first value of t such that $x(t)$ is used as a value of the dependent variable for fitting the model. t_1 is specified so that all models being compared within a run are trying to predict exactly the same set of data. t_1 must be large enough that $t_1 - T_p - (d-1)L \geq 1$ for all models fitted during the run. If the value of t_1 in the parameter file is too small it is reset automatically to the smallest possible for the run, so setting $t_1 = 1$ automatically selects the smallest possible t_1.

jforce, jper: If jforce= 1 the fitted model also includes periodic forcing of period jper. The model then has the form

$$x(t+T_p) \;=\; \alpha x(t) \;+\; f\Big(\,x(t), x(t-L), ...x(t-(d-1)L), cos(\omega t), sin(\omega t)\,\Big) + \text{noise.}$$

with $\omega = 2\pi/jper$. If jforce= 0 the cos and sin terms are omitted and the value of jper is irrelevant.

cgcv: Value of c for computing the modified GCV criterion V_c; cgcv= 2 is recommended.

The remaining parameters (marked with an asterisk) are explained in Section 5; these only affect the optimization subroutines, and can be ignored by the user unless something goes wrong.

4. AN EXAMPLE

Run LENNS with the parameter file lenns1.par, to estimate λ for the data file dmgm10.asc. These simulated data were generated by the discrete Mackey-Glass system

$$x_t = \left\{ a x_{t-1} + \frac{b x_{t-k}}{1+(x_{t-k})^j} \right\} * exp(0.1 * Z_t) \tag{10}$$

with $a = 0.2, b = 2.0, k = 2$, and $j = 6$, where $\{Z_t\}$ is Gaussian white noise (independent with zero mean and unit variance).

Results from 1 out of each 20 initial fits are written to lenns.tmp, which can be erased once the program has terminated. The output file lists the following variables:

Columns 1–3 :	L (time delay), d (embedding dimension), and k (# of units).
Columns 4–6 :	RMS error and the model selection criteria BIC and V_c.
Column 7:	Number of fitted parameters in the model
Columns 8:	Number of BFGS iterations
Column 9:	"info" from BFGS (0 if minimization succeeded, 1 if it failed)
Columns 10-11:	Estimated dominant LE, SVD and QR estimates

and it should look something like this (though not exactly):

```
@ /home/ellner/data/dmgm10.asc       Output from LENNS.FOR
Dependent variable data scaled to Std Dev == 1
Parameter values:
125   18  1  1          <-- nx,jt1,ltr,ifreq
  1    6   1   2.000     <-- kmin,kmax,kcrit,cgcv
  1   0.00   0.50        <-- T_p,alfa,scale
  0     12               <-- jforce, jper
0.100E-05   0.500E-09  <-- ftol1,ftol2
   800  10000     50    250  <-- itmax1,itmax2,maxstep1,maxstep2
output l,d,k,RMS err,BIC,GCV,# params,iters,info,LE1:SV&QR @
   1  1  1   0.985226   1.4915   1.1339  4    207  0   -4.897   -4.897
   1  1  1   0.985296   1.4915   1.1340  4    201  0   -4.600   -4.600
   1  1  1   0.985290   1.4915   1.1340  4    151  0   -4.626   -4.626
   1  1  1   0.985273   1.4915   1.1340  4    257  0   -4.696   -4.696
   1  1  1   0.985270   1.4915   1.1340  4    226  0   -4.710   -4.710
   1  1  1   0.985299   1.4915   1.1341  4     96  0   -4.588   -4.588
   1  1  1   0.985291   1.4915   1.1340  4    229  0   -4.619   -4.619
   1  1  1   0.985232   1.4915   1.1339  4    152  0   -4.875   -4.875
```

```
[about 1500 lines omitted]
   3  6  5    0.333966    1.2175    2.0431 41    5901  0     0.298     0.296
   3  6  5    0.341999    1.2413    2.1426 41   10001  0     0.457     0.412
   3  6  5    0.335903    1.2233    2.0669 41    9791  0     0.289     0.269
   3  6  5    0.334318    1.2186    2.0474 41   10001  0     0.353     0.347
   3  6  5    0.336277    1.2244    2.0715 41   10001  0     0.287     0.276
   3  6  5    0.343566    1.2459    2.1623 41   10001  0     0.291     0.277
   3  6  5    0.335355    1.2217    2.0601 41    2645  0     0.355     0.346
   3  6  5    0.334335    1.2187    2.0476 41    9269  0     0.293     0.280
   3  6  5    0.328236    1.2002    1.9736 41    5330  0     0.186     0.176
   3  6  5    0.325866    1.1930    1.9452 41    5042  0     0.226     0.223
@ending time=   714382763 elapsed time  116694@
```

The output file is long for two reasons. First, smoothing parameters are chosen by fitting a range of models with different choices of the time-delay L, embedding dimension d, and number of units k, and afterwards finding the winner (a possible shortcut is suggested below).

Second, fitting neural nets by nonlinear least squares is a numerical nightmare: the parameter space is full of local minima, and directions along which the RMS error has an asymptote rather than local minima (which can crash standard minimization routines). The solution in LENNS is: if you're shooting blind, shoot often. For each (L, d, k) combination, 100 parameter sets are generated at random (uniform distribution in a cube centered at 0), and the one with the lowest RMS is used as the initial point or iterative minimization of the RMS error. If minimization fails without satisfying the convergence criterion, a failure is declared (info= 1 on return from bfgsmin) and 100 new random parameter sets are drawn from the same cube. This is repeated 250 times with the cube growing geometrically from "too small" to "too big". The parameters and final RMS for the 250 trial fits are saved, and the 20 parameter sets with the lowest RMS are "polished" by continuing the minimization with a more stringent convergence tolerance.

The output file gives the results for all 20 polished fits, to let the user determine whether there are multiple local minima with similar RMS's but different LE's. Multiple minima with significantly different LE's mostly occur at specifications more complex than those selected by BIC or GCV. Problems at the BIC- or GCV-preferred specification are a sign that the neural net model doesn't fit the data well.

The output file needs to be summarized, and we suggest two graphical summaries:

1. A scatterplot of estimated LE vs. V_c or BIC (Fig. 1a). For each (L,d,k) combination in the output file, the plot shows LE and V_c for the single best fit. It helps to use different symbols for each value of L, and it is essential to identify points corresponding to the

optimal k (lowest V_c) for each (L,d) combination, because networks that are too simple or too complex will underestimate or overestimate λ, respectively. In Figure 1a, V_c clearly picks the correct time delay (L= 1) and the better fits all give fairly accurate estimates of λ (the true value is about 0.14).

2. A plot of estimated λ vs. the number of lags (d), for the L of the single best fit (Fig. 1b). In theory λ should be constant as the number of lags in the model is increased above the minimum number needed for a valid reconstruction. For each d, we plot the single best fit (over all fits at all values of k) and the mean±1 standard deviation for the 10 best polished fits at the k of the best fit. Estimated λ's that don't become roughly constant as d is increased are a sign of trouble, e.g., data that are too high dimensional for the "best" fit to be any good. A large standard deviation indicates that estimates are unreliable due to the numerical inaccuracy of parameter estimates. In Fig. 1b, there is a definite plateau for d$\geq$ 2 and the numerical inaccuracy is negligibly small.

Figure 2 summarizes the corresponding results for a 4th-order linear approximation to the discrete Mackey-Glass system,

$$y_t = .06y_{t-1} - .66y_{t-2} - .12y_{t-4} + \sigma Z_t \tag{11}$$

where $y_t = ln\, x_t$, $\sigma = 0.36$, and the $\{Z_t\}$ are Gaussian white noise (data file ardmg4.asc). The coefficients in (11) were chosen by maximum likelihood fitting to logged "data" from dmgm10.asc . The true value of λ for this system is -0.46, and the estimated value is about -0.3 . This is qualitatively OK – the system is identified as nonchaotic – but it illustrates the desirability of having confidence intervals instead of point estimates (which we hope to provide soon).

Of course with real data the results can be less tidy (Figure 3). The results for prediction time $T_p = 1$ (Fig. 3a) favor L= 1, and the estimated λ is stable with increasing embedding dimension (Fig. 3b). However this data set (pred11.asc) has a strong autocorrelation at lag 1, so linear extrapolation based on the last two lags would be a good short-term predictor even if the dynamics are nonlinear (indeed the lowest V_c occurs at L= 1, d= 2). To check for this possibility we re-estimated λ with $T_p = 2$ (Fig. 3c), and longer time delays were favored as expected, L= 2 or 4. For these L's there is no real plateau as d increases (Fig. 3d,e), so while it seems safe to say that $\lambda < 0$, it would be difficult to choose a single estimate.

In the discrete Mackey-Glass example, the fits for L= 2 and 3 were wastes of CPU time. Given enough data it is often possible to identify one or two "good" L and d values by the ordinary cross-validation (OCV) method of Cheng and Tong (1992), with the

modifications discussed above to account for autocorrelations. OCV is much quicker than running LENNS for multiple L and d values. The program cvmbed.for is an implementation of the OCV method; the parameter file cvmbed.par follows the same conventions as lenns.par.

For the noisy Mackey-Glass system, 125 data points are not enough to pick out the correct time delay (L= 1), and 250 data points are barely adequate (Table 1); even with 250 data points the correct embedding dimension (d= 2) could not be picked with confidence. Thus for this example the Lyapunov exponent is easier to estimate than the embedding, in terms of the amount of data required, because the estimate of λ is robust to overestimating the dimension. However the Lyapunov exponent is much harder to estimate in terms of the CPU time required, by several orders of magnitude. Which brings us to....

b 5. FINE-TUNING THE OPTIMIZATION

The optimization algorithm is standard BFGS, with the BHHH steplength algorithm (Berndt et al. 1974), and the convergence criterion suggested by Gill et al. (1981) for unconstrained minimization. So about those * parameters:

scale: sets the range of cube sizes for the initial conditions. The data are re-scaled inside LENNS to unit variance, and then scale= 0.5 seems to be a good choice in that the best fits almost always come from intermediate cube sizes.

tol1, tol2: these set the convergence tolerances for the 250 initial fits and the 20 "polished" fits, respectively.

itmax1, itmax2: maximum number of BFGS iterations for the 250 & 20 fits.

maxstep1, maxstep2: maximum number of BHHH iterations per BFGS iteration during the 250 & 20 fits.

ibrent, fltol: If ibrent= 1, then for the polished fits, BFGS will attempt an "exact" line-minimization using Brent's quadratic interpolation method to choose the step length, with the BHHH steplength as the initial guess. fltol is the convergence criterion for the line minimization (accurate to ±10% if fltol= 0.1, etc.). Try setting ibrent= 1 if the "polished" are good but not great (too much difference between fits, or too many failures to converge). Brent's method is less robust than BHHH, but it may improve the final accuracy if the initial parameters are near to a minimum.

BFGS/BHHH is the most efficient and reliable public-domain method we found for

fitting neural net models by least squares, but equally good results are obtained somewhat faster with the (copyrighted) package NPSOL (Gill et al. 1986). The same algorithm is available as NAG routine *e04ucf*. LENNS is written so that calls to NPSOL or *e04ucf* can easily be substituted for the calls to bfgsfm in lenns.for. The speedup is about 25%. If you switch to NPSOL we recommend the parameter settings

MAJOR ITERATION LIMIT= 400 (initial fits), 1200 (polished fits)

OPTIMALITY TOLERANCE= 10^{-5} (initial fits), 10^{-8} (polished fits)

and otherwise the NPSOL defaults are OK. Subroutine MAKEOPT (in util_le.for) can be called to set up the options files for NPSOL.

Acknowledgements. Our ideas and programs owe a great deal to our discussions with colleagues, especially Alfredo Asceoti, Matthew J. Saltzman, James Theiler, and Peter Turchin. Parts of the code are based on programs written by Dan McCaffrey as part of his dissertation research. We thank the North Carolina Supercomputing Center for time on their Cray. This research was supported in part by NSF grants DMS-8715756 and SES-8808015, and North Carolina Agricultural Experiment Station Project NCO-6134.

BIBLIOGRAPHY

Abarbanel, H.D.I. 1992. Lyapunov exponents in chaotic systems: Their importance and their evaluation using observed data. Modern Physics Letters B (in press).

Barron, A.R. 1991a. Complexity regularization with application to artificial neural nets. pp. 561-576 in: G. Roussas (ed.) Nonparametric Function Estimation and Related Topics, Kluwer Academic Publishers, the Netherlands.

Barron, A.R. 1991b. Approximation and estimation bounds for artificial neural networks. PP. 243-249 in: Proc. Fourth Annual Workshop on Computational Learning Theory, Univ. Calif. Santa Cruz, August 5-7 1991. Morgan Kaufmann Publ, San Mateo CA.

Berndt, E.K., G.H. Hall, R.E. Hall, and J.A. Hausman. 1974. Estimation and inference in nonlinear structural models. Annals of Economics and Social Measurement, 3/4:653-665.

Brown, R., P. Bryant, and H.D.I. Abarbanel. 1991. Computing the Lyapunov spectrum of a dynamical system from an observed time series. Physical Review A 43, 2787-2805.

Burnett, T. 1964. An acarine predator-prey population infecting stored products. Canadian J. Zoology 42: 655-673.

Casdagli, M. 1989. Nonlinear prediction of chaotic time series. Physica D35, 335-356

Casdagli, M. 1992a. A dynamical systems approach to modeling input-output systems. pp. 265-281 in: M. Casdagli and S. Eubank (eds.) Nonlinear Modeling and Forecasting. SFI Studies in the Sciences of Complexity Proc. Vol. XII. Addison-Wesley, NY.

Casdagli, M. 1992b. Chaos and deterministic versus stochastic modeling. J. Royal Statistical Society B, 303-328.

Casdagli, M. and S. Eubank. 1992. Nonlinear Modeling and Forecasting. Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XII. Addison-Wesley, New York.

Cheng, B. and H. Tong. 1992. On consistent non-parametric order determination and chaos. Journal of the Royal Statistical Society Series B, 54:427-450.

Crutchfield, J.P., J.D. Farmer, and B.A. Huberman. 1982. Fluctuations and simple chaotic dynamics. Physics Reports 92, 45-82.

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems, 2:303-314.

Eckmann, J.-P. and D. Ruelle. 1985. Ergodic theory of chaos and strange attractors. Reviews of Modern Physics, 57(3):617-656.

Eckmann, J.-P., S.O. Kamphorst, D. Ruelle, and S. Ciliberto. 1986. Liapunov exponents from time series. Physical Review A, 34:4971-4979.

Ellner, S., A.R. Gallant, D. McCaffrey, and D. Nychka. 1991. Convergence rates and data requirements for Jacobian-based estimates of Lyapunov exponents from data. Physics Letters 153, 357-363.

Friedman, J. and B.W. Silverman. 1989. Flexible parsimonious smoothing and additive modeling. Technometrics 31:3-21.

Friedman, J. 1990. Multivariate adaptive regression splines. Annals of Statistics 19:1-67.

A.R. Gallant and H.L. White. 1991. On learning the derivatives of an unknown mapping with multilayer feedforward networks. Neural Networks 5: 129-138.

Gill, P.E., W. Murray, and M.H. Wright. 1981. Practical Optimization. Academic Press, London and New York.

Gill, P.E., W. Murray, M.A. Saunders, and M.H. Wright. 1986. Users Guide for NPSOL (Version 4.0): A Fortran Package for Nonlinear Programming. Technical Report SOL 86-2, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford CA 94305.

Granger, C.W.J. and T. Teräsvirta. 1992. Experiments in modeling nonlinear relationships between time series. pp. 199-226 in M. Casdagli and S. Eubank (eds.), Nonlinear Modeling and Forecasting. Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XII. Addison-Wesley, New York.

Guckenhiemer, J. 1982. Noise in chaotic systems. Nature 298: 358-361.

T. Hastie and R. Tibshirani. 1990. Generalized additive models, Chapman and Hall, Ltd., London, England.

R. Hecht-Nielsen. 1989. Theory of the backpropagation neural network. International Joint Conference on Neural Networks, San Diego, California, SOS Printing.

K. Hornick, M. Stinchcombe, and H. White. 1989. Multilayer feedforward networks are universal approximators. Department of Economics, University of California, San Diego, California (discussion paper).

IJCNN. 1989. International Joint Conference on Neural Networks, San Diego, California, SOS Printing.

Kifer, Y. 1986. Ergodic Theory of Random Transformations. Birkhäuser, Boston, Massachusetts.

McC 991. Estimating Lyapunov exponents with nonparametric regression and convergence rates for feed forward single hidden layer networks. Ph.D. thesis, Department of Statistics, North Carolina State University, Raleigh, North Carolina.

McC S. Ellner, D.W. Nychka, and A.R. Gallant. 1992. Estimating the Lyapunov exponent of a chaotic system with nonlinear regression. J. Amer. Stat. Assoc. 87: 682-695.

McCulloch, W.S. and W. Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5:115-143.

Nychka, D.W., S. Ellner, D. McCaffrey, and A.R. Gallant. 1992. Finding chaos in noisy systems. Journal of the Royal Statistical Society Ser. B 54: 399-426.

Sayers, C. 1990. Chaos and the business cycle. pp. 115-125 in: S. Krasner (ed.), The Ubiquity of Chaos. AAAS, Washington DC.

Sugihara, G. and R.M. May. 1990. Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. Nature 344, 734-741.

Theiler, J. 1986. Spurious dimensions from correlation algorithms applied to limited time-series data.

Physical Review Letters 34: 2427-2432.

Wahba, G. 1990. Spline Models for Observational Data, SIAM, Philadelphia.

White, H. 1989. Some asymptotic results for learning in single layer feedforward network models. J. Amer. Stat. Assoc. 84:1003-1013.

Wolf, A., J.B. Swift, H.L. Swinney, and J.A. Vastano. 1985. Determining Lyapunov exponents from a time series. Physica 16D, 285-315.

Table 1. Results from ordinary cross-validation to choose embedding parameters, based on 125 and 250 data points from dmgm10.asc, using the method of Cheng and Tong (1992) with a kernel regression model (rational approximation to a Gaussian kernel). Window length 18, W= 3, data log transformed and scaled to unit variance. CV, the cross-validated estimate of RMS prediction error, was calculated for L= 1 to 3, prediction time T_p = 1 to 4 time steps ahead, and embedding dimensions (# lags) d= 1 to 6. CV_{opt} is the minimum value of CV (optimized over bandwidth and embedding dimension), and d_{opt} is the embedding dimension at which CV_{opt} occurred.

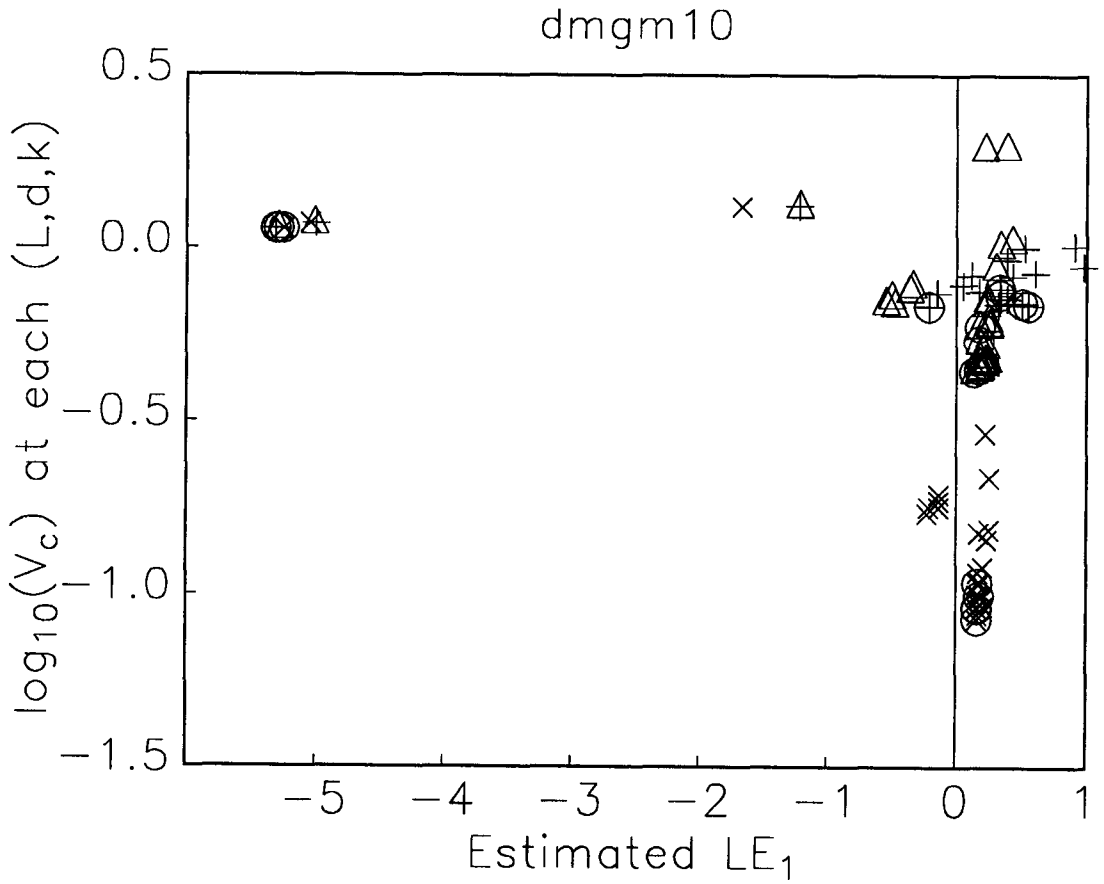
T_p	L	125 points		250 points	
		d_{opt}	CV_{opt}	d_{opt}	CV_{opt}
1	1	2	.38	2	.31
	2	2	.91	4	.87
	3	3	.71	3	.63
2	1	1	.33	2	.30
	2	1	.33	1	.33
	3	1	.33	2	.33
3	1	2	.66	2	.58
	2	2	.88	3	.90
	3	2	.77	2	.71
4	1	2	.70	2	.60
	2	1	.72	1	.70
	3	2	.69	2	.64

FIGURE LEGENDS

Figure 1. Summary of results in dmgm10.le1, for 125 simulated data points from the noisy Mackey-Glass system (10). (a) Scatterplot of V_c with $c = 2$ vs. estimated Lyapunov exponent for the best fit at each (L, d, k) combination. Time delays $L = 1(\times)$, L=2(+), $L = 3(\Delta)$; circles indicate the best fit at each (L, d) combination. (b) Plot of estimated LE vs. embedding dimension (#lags) for $L = 1$. The plot shows the LE at the single best fit (□), and the mean (Δ) $\pm$ 1 standard deviation (dots) for the 10 best fits at the best value of k.

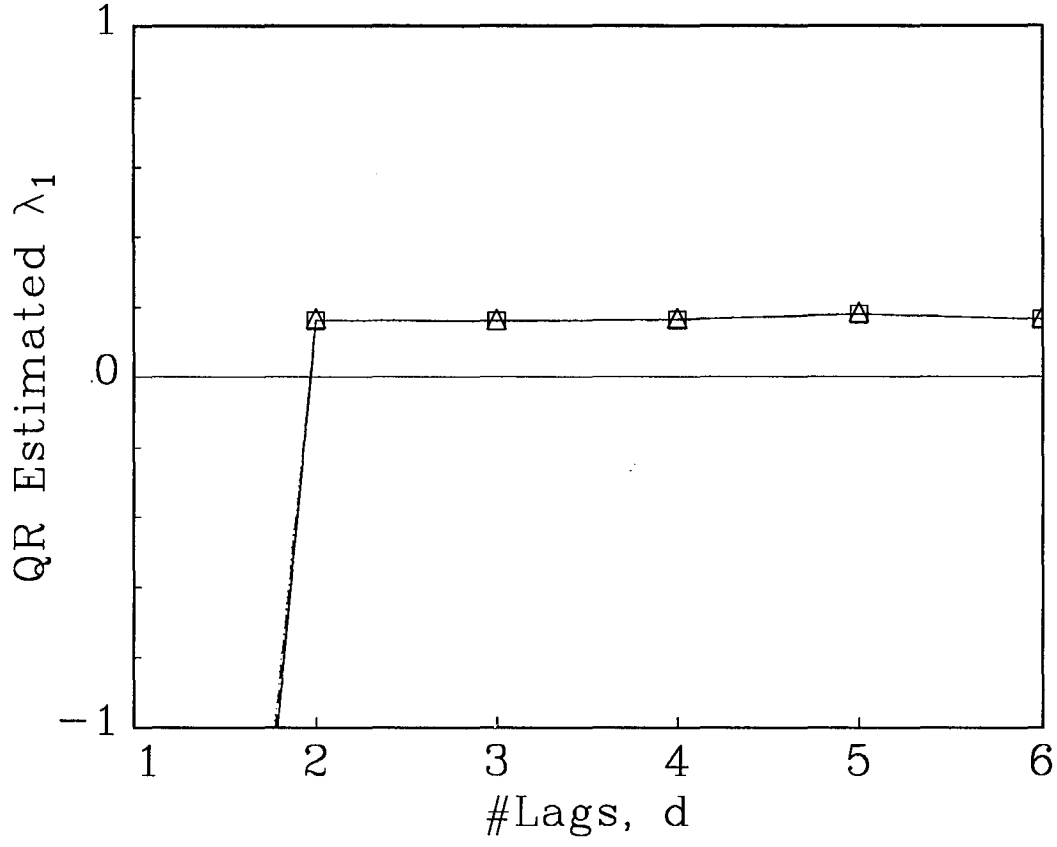
Figure 2. As in Figure 1, for 125 simulated data points from the fourth-order linear approximation (11) to the noisy Mackey-Glass system.

Figure 3. As in Figure 1 for an experimental data set on predator-prey population oscillations, 98 values of population abundance of the predator mite *Blattisocius dendriticus* (Burnett 1964, Fig. 7b). (a), (b): prediction time $T_p = 1$. (c),(d),(e): prediction time $T_p = 2$. Symbols as in Fig.1,except ∇ indicates $L = 4$ in (a) and (c).

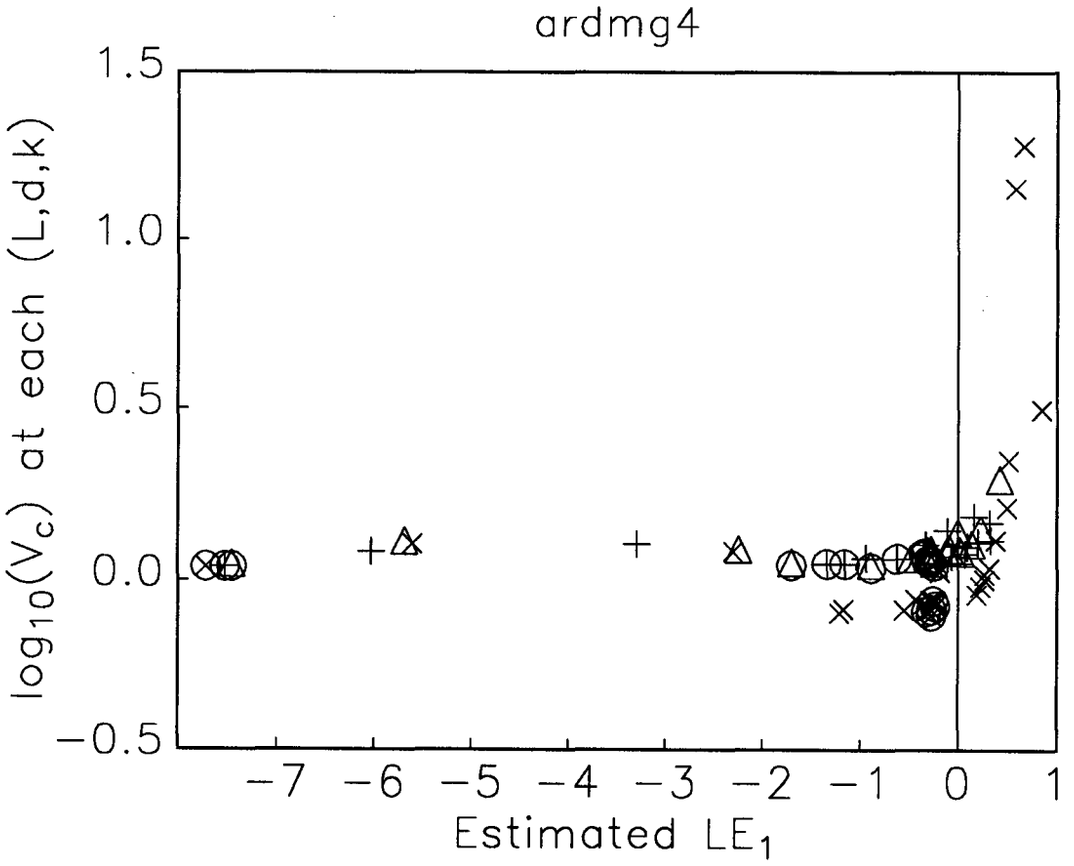


1a

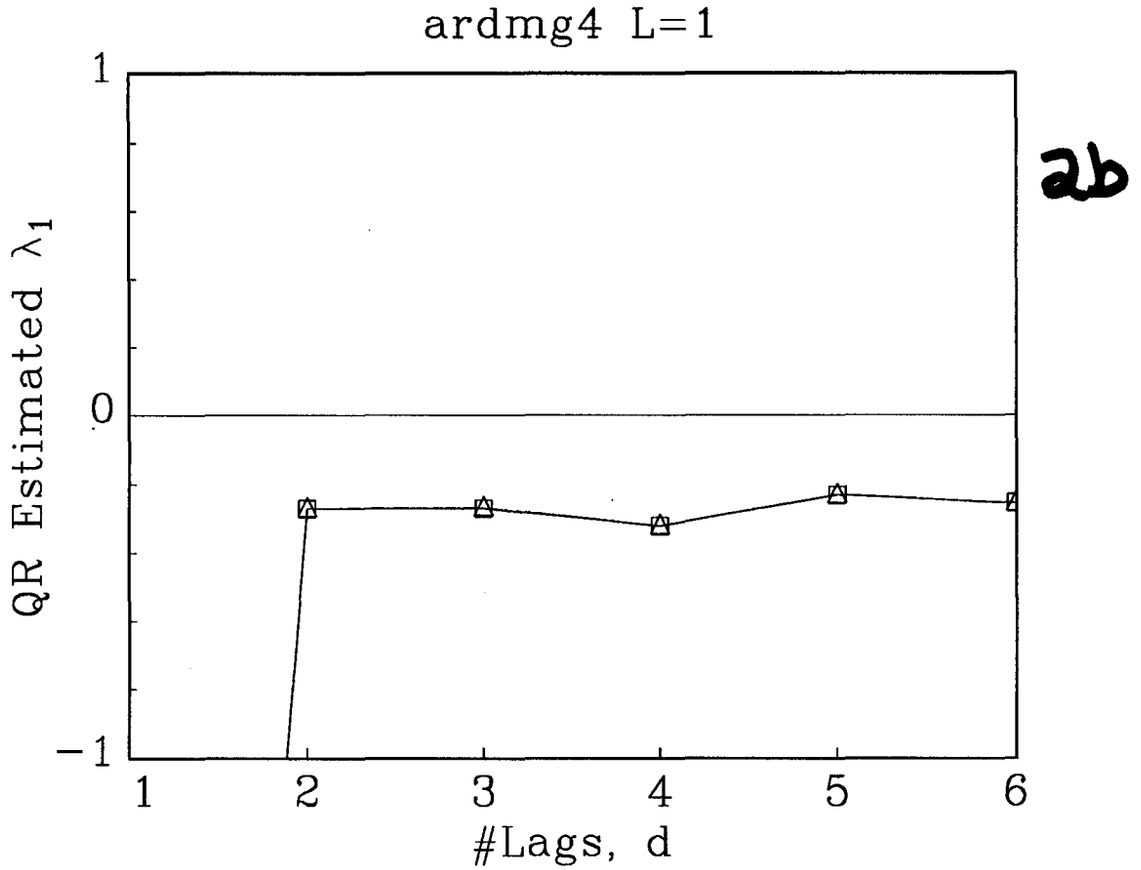
dmgm10 L=1



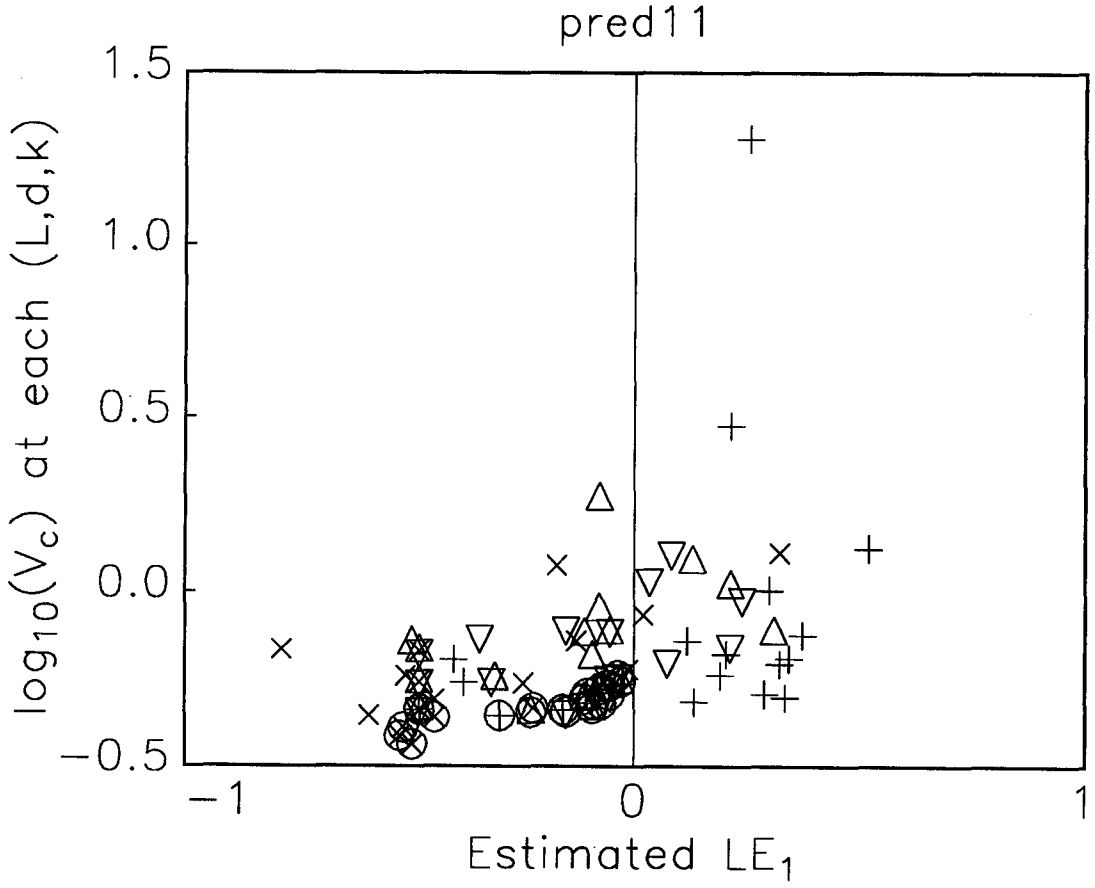
1b


ardmg4
1.5
1.0
0.5
0.0
−0.5
log10(Vc) at each (L,d,k)
−7 −6 −5 −4 −3 −2 −1 0 1
Estimated LE1

2a

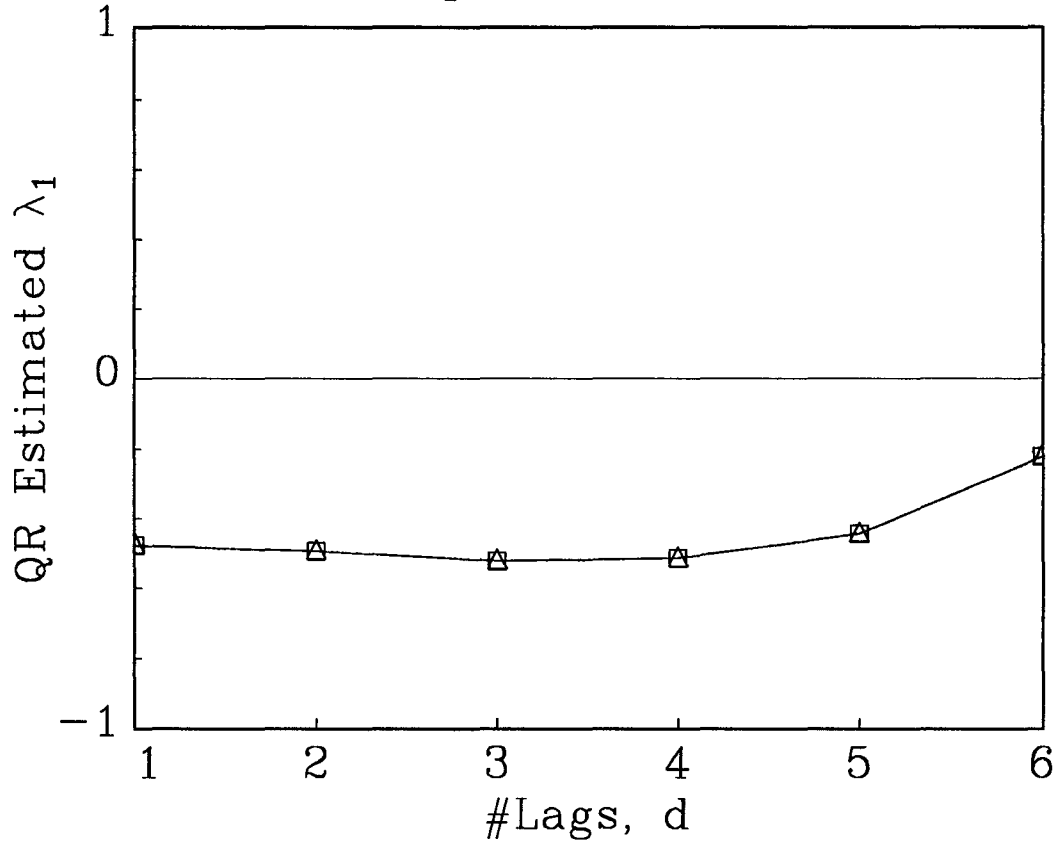

ardmg4 L=1
1
0
−1
QR Estimated λ1
1 2 3 4 5 6
#Lags, d

2b

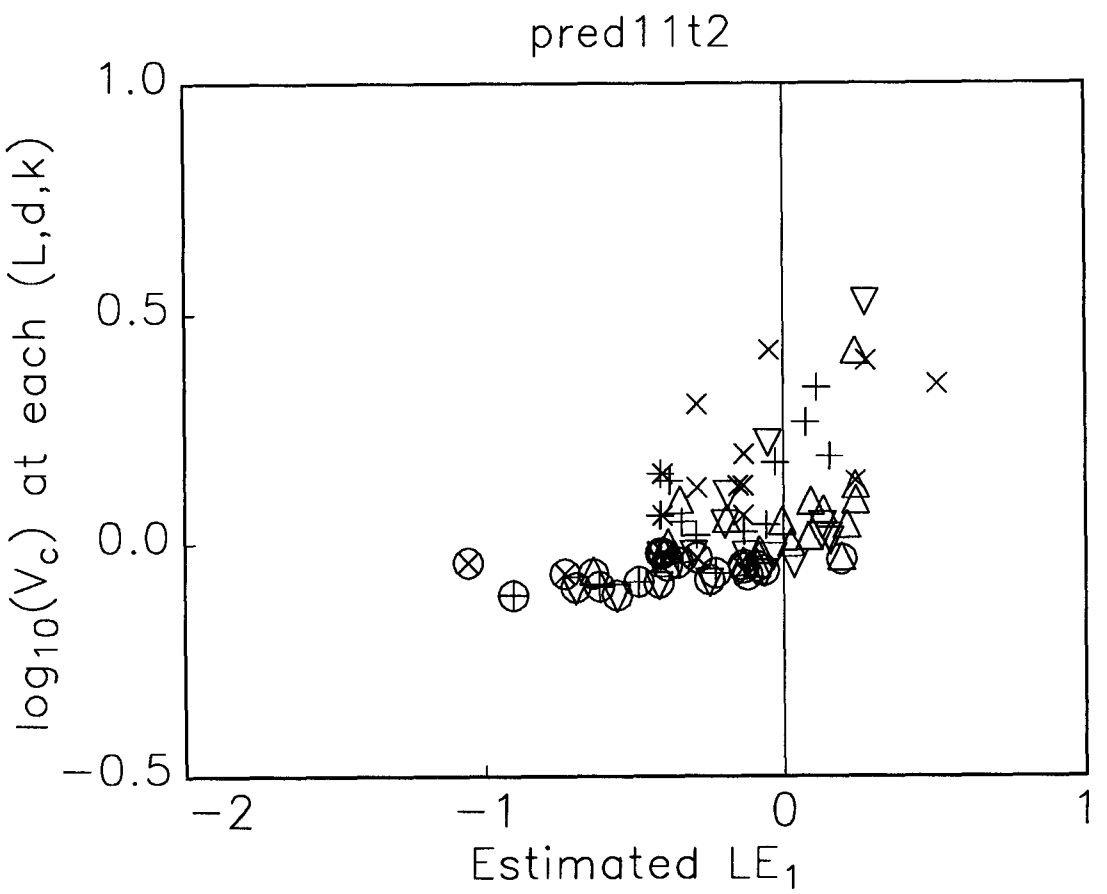


3a

pred11 L=1

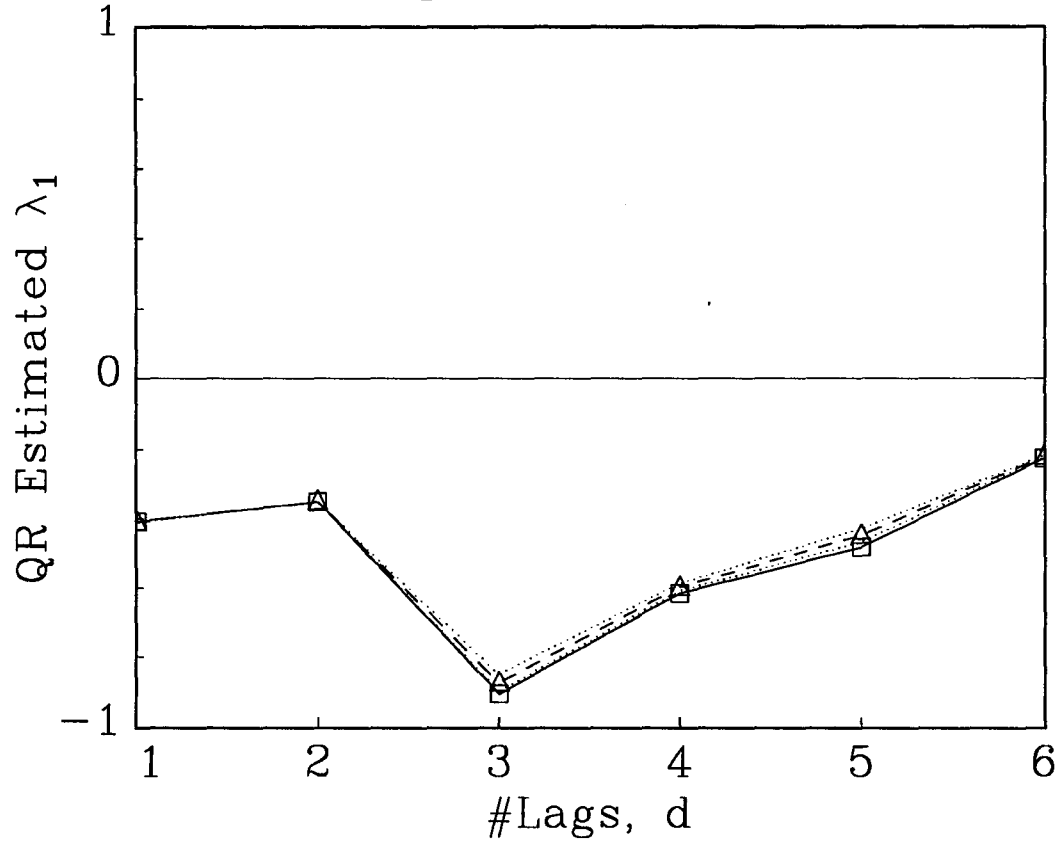


3b

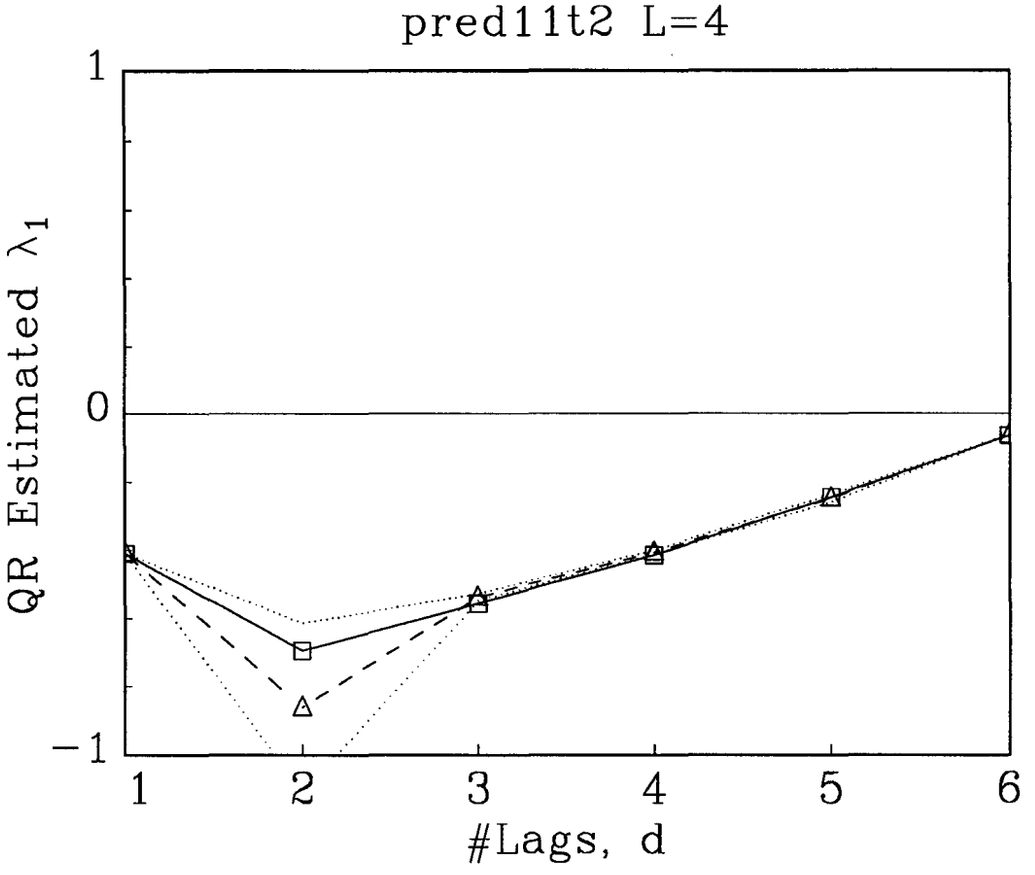


3c

pred11t2 L=2



3d



3e